ORNL/TM-13006

Computer Science and Mathematics Division

Mathematical Sciences Section

MESSAGE-PASSING PERFORMANCE OF VARIOUS COMPUTERS

Jack J. Dongarra and Thomas H. Dunigan

Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367
dongarra@cs.utk.edu thd@ornl.gov

Date Published: February 1996

Research was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Lockheed Martin Energy Research Corp.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-96OR22464

Contents

1 Introduction and Motivation 1
1.1 The Rise of the Microprocessor 1
1.2 Communications and Parallel Processing Systems 1
2 Message Passing 2
2.1 Programming Model 2
2.2 Measurement Methodology 3
2.3 Latency and Bandwidth 4
3 Computation and Communication 7
3.1 Performance 7
3.2 The LINPACK Benchmark 9
4 Future comparisons 11
4.1 Rules for Running the Tests 11
4.2 Obtaining the Software 11
5 References 12
A Appendix: Machine Configurations for Echo Tests 13

MESSAGE-PASSING PERFORMANCE OF VARIOUS COMPUTERS

Jack J. Dongarra and Thomas H. Dunigan

Abstract

This report compares the performance of different computer systems for basic message passing. Latency and bandwidth are measured on Convex, Cray, IBM, Intel, KSR, Meiko, nCUBE, NEC, SGI, and TMC multiprocessors. Communication performance is contrasted with the computational power of each system. The comparison includes both shared and distributed memory computers as well as networked workstation clusters.

1. Introduction and Motivation

1.1. The Rise of the Microprocessor

The past decade has been one of the most exciting periods in computer development that the world has ever experienced. Performance improvements have been dramatic, and that trend promises to continue for the next several years.

In particular, microprocessor technology has changed rapidly. Microprocessors have become smaller, denser, and more powerful. Indeed, microprocessors have made such progress that, if cars had made equal progress since the day they were invented, we would now be able to buy a car for a few dollars, drive it across the country in a few minutes, and not worry about parking because the car would fit into one's pocket. The result is that the vendors of high-performance computing have turned to RISC microprocessors for performance.

Collections of these processors are interconnected by hardware and software to attack various applications. The physical interconnection of these processors may be contained in one or more cabinets as part of a multiprocessor, or the processors may be standalone workstations dispersed across a building or campus interconnected by a local area network. The effectiveness of using a collection of processors to solve a particular application is constrained by the amount of parallelism in the application, compiler technology, message passing software, amount of memory, and by the speed of the processors and of the interconnecting network.

1.2. Communications and Parallel Processing Systems

This report compares the results of a set of benchmarks for measuring communication time on a number of NUMA computers ranging from a collection of workstations using PVM [5] to machines like the IBM SP-2 and the Cray T3D using their native communication library, MPI [4], or PVM. We are interested in the communication performance for a number of reasons. First, our main interest is to obtain fundamental parameters on a given hardware platform to help in building models of execution. Second, performance data can be used to compare machines and help to evaluate new machines and architectures as they become available.

The following section describes the critical parameters in evaluating message passing systems. The techniques to measure these parameters are described. In section 3, the message passing performance of several multiprocessors and networks are presented. Communication and computational performance are contrasted. Section 4 provides details for obtaining the test software.

2. Message Passing

2.1. Programming Model

Processes of a parallel application distributed over a collection of processors must communicate problem parameters and results. In distributed memory multiprocessors or workstations on a network, the information is typically communicated with explicit message-passing subroutine calls. To send data to another process, a subroutine is usually provided that requires a destination address, message, and message length. The receiving process usually provides a buffer, a maximum length, and the senders address. The programming model is often extended to include both synchronous and asynchronous communication, group communication (broadcast and multicast), and aggregate operations (e.g., global sum).

Message passing performance is usually measured in units of time or bandwidth (bytes per second). In this report, we choose time as the measure of performance for sending a small message. The time for a small, or zero length, message is usually bounded by the speed of the signal through the media (latency) and any software overhead in sending/receiving the message. Small message times are important in synchronization and determining optimal granularity of parallelism. For large messages, bandwidth is the bounding metric, usually approaching the maximum bandwidth of the media. Choosing two numbers to represent the performance of a network can be misleading, so the reader is encouraged to plot communication time as function of message length to compare and understand the behavior of message passing systems.

Message passing time is usually a linear function of message size for two processors that are directly connected. For more complicated networks, a per-hop delay may increase the message passing time. Message-passing time, t_n, can be modeled as

$$t_n = \alpha + \beta n + (h-1)\gamma$$

with a start-up time, α, a per-byte cost, β, and a per-hop delay, γ, where n is the number of bytes per message and h the number of hops a message must travel. On most current message-passing multiprocessors the per-hop delay is negligible due to "worm-hole" routing techniques and the small diameter of the communication network [3]. The results reported in this report reflect nearest-neighbor communication. A linear least-squares fit can be used to calculate α and β from experimental data of message-passing times versus message length. The start-up time, α, may be slightly different than the zero-length time, and $1/\beta$ should be asymptotic bandwidth. The message length at which half the maximum bandwidth is achieved, $n_{1/2}$, is another metric of interest and is equal

to α/β [6]. As with any metric that is a ratio, any notion of "goodness" or "optimality" of $n_{1/2}$ should only be considered in the context of the underlying metrics α and β.

There are a number of factors that can affect the message passing performance. The number of times the message has to be copied or touched (e.g., checksums) is probably most influential and obviously a function of message size. The vendor may provide hints as to how to reduce message copies, for example, by posting the receive before the send. Second order effects of message size may also affect performance. Message lengths that are powers of two or cache-line size may provide better performance than shorter lengths. Buffer alignment on word, cache-line, or page may also affect performance. For small messages, context-switch times may contribute to delays. Touching all the pages of the buffers can reduce virtual memory effects. For shared media, contention may also affect performance. There also may be some first-time effects that can be identified or eliminated by performing some "warm up" tests before collecting performance data.

There are of course other parameters of a message-passing system that may affect performance for given applications. The aggregate bandwidth of the network, the amount of concurrency, reliability, scalability, and congestion management may be issues.

2.2. Measurement Methodology

To measure latency and bandwidth, we use a simple echo test between two adjacent nodes. A receiving node simply echos back whatever it is sent, and the sending node measures round-trip time. Times are collected for some number of repetitions (100 to 1000) over various messages sizes (0 to 1,000,000 bytes). Times can be collected outside the repetition loop as illustrated in Figure 2.1. If the system has high resolution timers then a more detailed analyses can be made by timing each send-receive pair. The time for each send-receive is saved in a vector and printed at the end of the test. For small message sizes, clock resolution may not be adequate, and clock jitter from time-sharing interrupts in the underlying OS may be observed. Unidirectional transfer time, or latency, is calculated as the minimum send-receive time (divided by two) for zero-length messages. Data rate, or bandwidth, is calculated from the number of bytes sent divided by half the round-trip time.

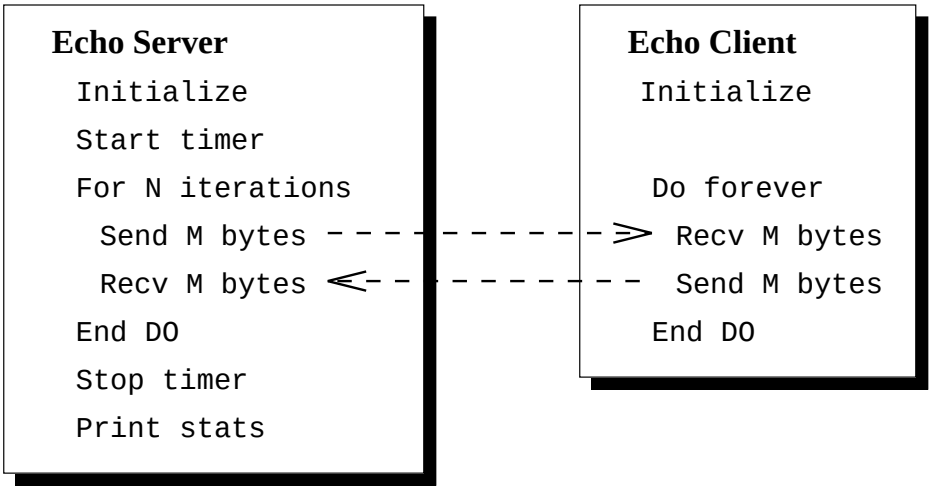

Figure 2.1: *Echo test pseudo-code.*

2.3. Latency and Bandwidth

Latency and bandwidth were measured on a number of different multiprocessors. Each architecture is briefly summarized in Appendix A. Table 2.3 shows the measured latency, bandwidth, and $n_{1/2}$ for nearest neighbor communication. The table also includes the peak bandwidth as stated by the vendor. For comparison, typical data rates and latencies are reported for several local area network technologies.

Table 2.1: *Multiprocessor Latency and Bandwidth.*

Machine	OS	Latency $n = 0$ (μs)	Bandwidth $n = 10^6$ (MB/s)	$n_{1/2}$ bytes	Theoretical Bandwidth (MB/s)
Convex SPP1000 (PVM)	SPP-UX 3.0.4.1	76	11	1000	250
Convex SPP1000 (sm 1-n)	SPP-UX 3.0.4.1	2.5	82	1000	250
Convex SPP1000 (sm m-n)	SPP-UX 3.0.4.1	12	59	1000	250
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	63	15	1000	250
Convex SPP1200 (sm 1-n)	SPP-UX 3.0.4.1	2.2	92	1000	250
Convex SPP1200 (sm m-n)	SPP-UX 3.0.4.1	11	71	1000	250
Cray T3D (sm)	MAX 1.2.0.2	3	128	363	300
Cray T3D (PVM)	MAX 1.2.0.2	21	27	1502	300
Intel Paragon	OSF 1.0.4	29	154	7236	175
Intel Paragon	SUNMOS 1.6.2	25	171	5856	175
Intel Delta	NX 3.3.10	77	8	900	22
Intel iPSC/860	NX 3.3.2	65	3	340	3
Intel iPSC/2	NX 3.3.2	370	2.8	1742	3
IBM SP-1	MPL	270	7	1904	40
IBM SP-2	MPI	35	35	3263	40
KSR-1	OSF R1.2.2	73	8	635	32
Meiko CS2 (sm)	Solaris 2.3	11	40	285	50
Meiko CS2	Solaris 2.3	83	43	3559	50
nCUBE 2	Vertex 2.0	154	1.7	333	2.5
nCUBE 1	Vertex 2.3	384	0.4	148	1
NEC Cenju-3	Env. Rel 1.5d	40	13	900	40
NEC Cenju-3 (sm)	Env. Rel 1.5d	34	25	400	40
SGI	IRIX 6.1	10	64	799	1200
TMC CM-5	CMMD 2.0	95	9	962	10
Ethernet	TCP/IP	500	0.9		1.2
FDDI	TCP/IP	900	9.7		12
ATM-100	TCP/IP	900	3.5		12

Figure 2.2 details the message-passing times of various multiprocessors over a range of message sizes. For small messages, the fixed overhead and latency dominate transfer time. For large message, the transfer time rises linearly with message size. Figure 2.3 illustrates the asymptotic behavior of bandwidth for large message sizes. It is possible to reduce latency on the shared-memory architectures by using shared-memory copy operations. These operations usually involve only one-processor and assume that the message is ready to be retrieved on the other processor. Figure 2.4 compares the message transfer times for the shared-memory *get* and explicit message passing for the Cray T3D, Meiko, and NEC. Current research in "active messages" is seeking ways to reduce message-passing overhead by eliminating context switches and message copying. Finally, Figure 2.5 graphically summarizes the communication performance of the various multiprocessors in a two-dimensional message-passing metric space. The upper-left region is the high performance area, lower performance and LAN networks occupy the lower performance region in the lower right.

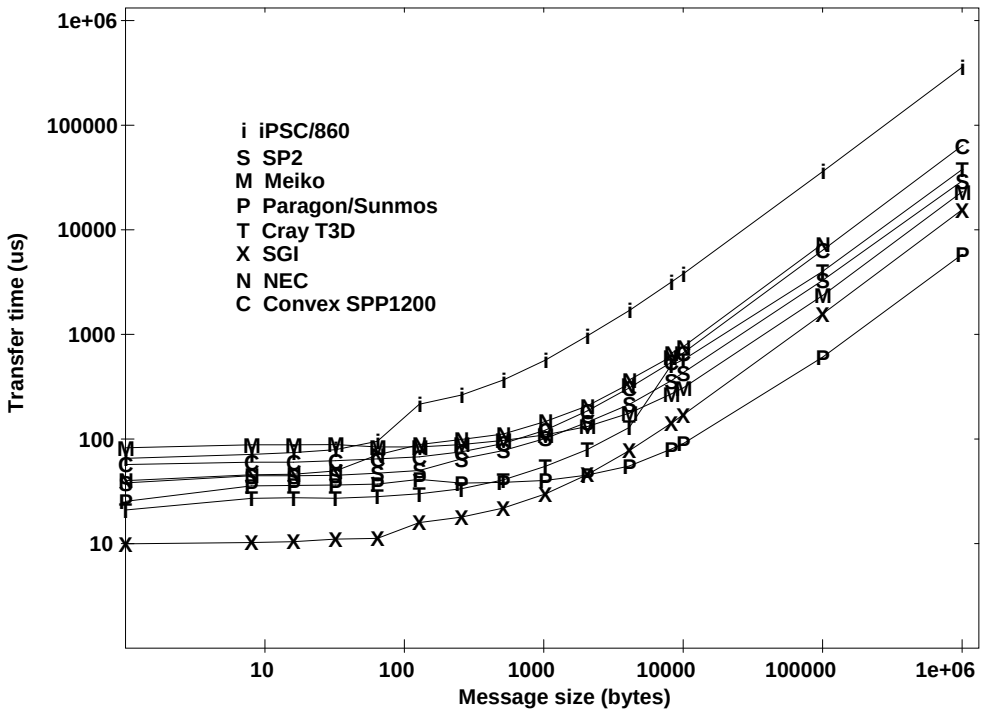

Figure 2.2: *Message-passing transfer time in microseconds for various multiprocessors and messages sizes.*

Since clusters of workstations on a network are often used as a virtual parallel machine, it is interesting to compare latency and bandwidths for various local area networks. Most communications over local area networks is done with the

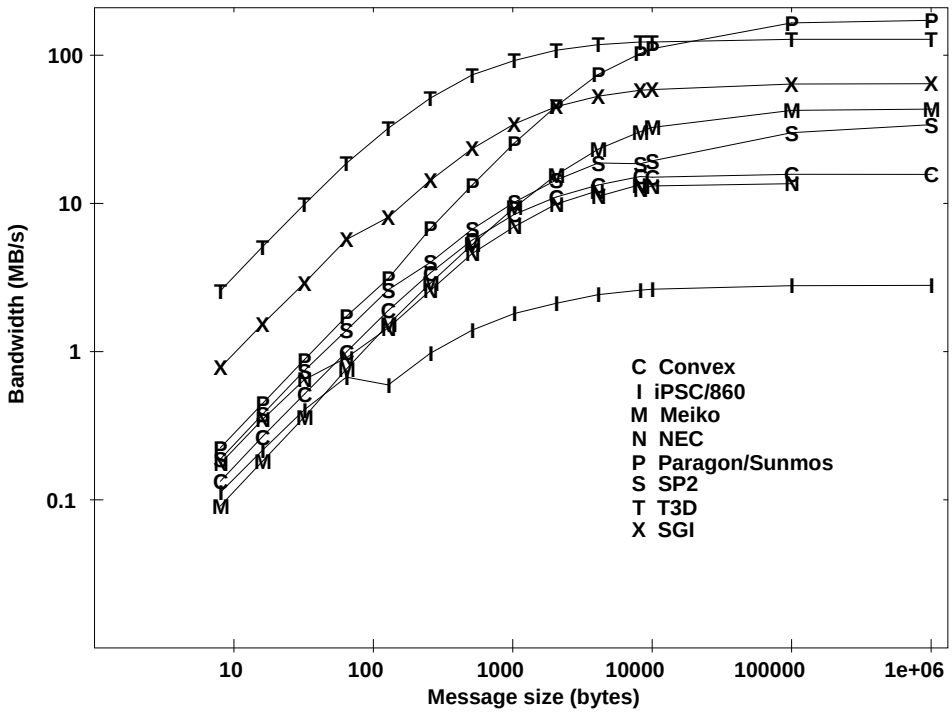

Figure 2.3: *Message-passing bandwidth in megabytes/second for various multiprocessors and messages sizes.*

TCP/IP protocols, though proprietary API's may exist. We measured latency for small messages using a UDP echo test. TCP bandwidth was measured at the receiver with the *ttcp* program using 50,000 byte messages and 50,000 byte window sizes. Some newer operating systems support even larger window sizes, which could provide higher bandwidths. Most high-end workstations can transmit network data at or near media data rates (e.g., 12 MB/second for FDDI). Data rates of 73 MB/second for UDP have been reported between Crays on HiPPI (and even over a wide-area using seven OC3's) [1]. Latency and bandwidth will depend as much on the efficiency of the TCP/IP implementation as on the network interface hardware and media. As with multiprocessors, the number of times the message is touched is a critical parameter as is context-switch time. Latencies for local area networks (Ethernet, FDDI, ATM, HiPPI) are typically on the order of 500 μs. For wide-area networks, latency is usually dominated by distance (speed of light) and is on the order of tens of milliseconds.

Message passing is often the limiting factor in performance of a parallel computer, so it is insightful to compare communication performance with computational performance for various architectures. The relative speed of computation and communication can be used in choosing the granularity of parallelism in im-

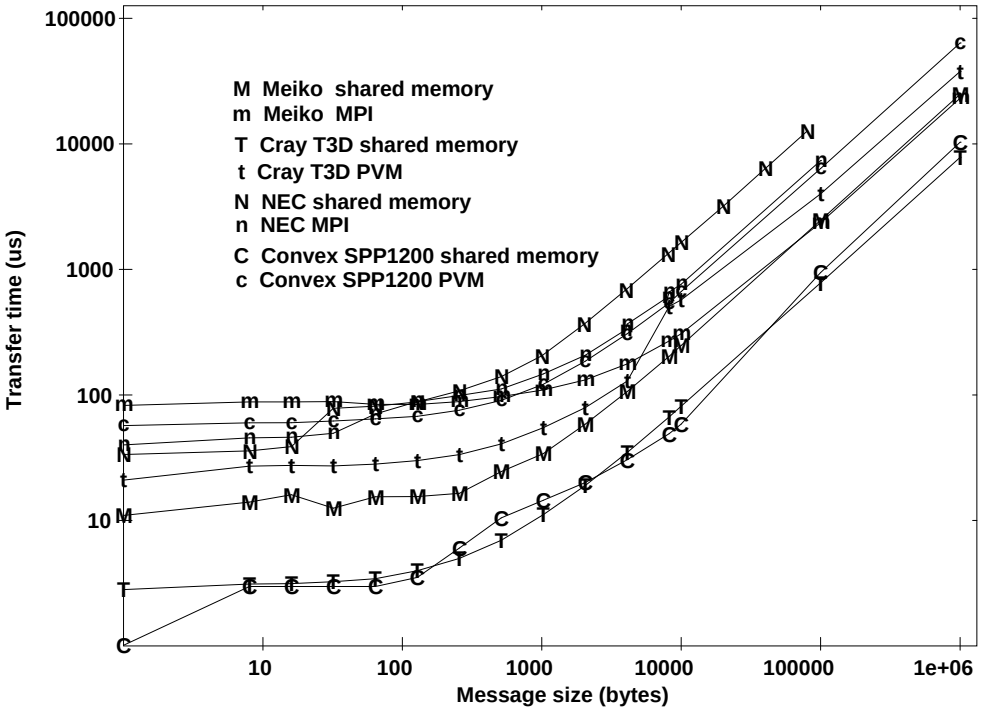

Figure 2.4: *Transfer time in microseconds for both shared-memory operations and explicit message passing.*

plementing a given application, or in optimizing the movement of data between processors or between levels of the memory hierarchy.

3. Computation and Communication

3.1. Performance

The performance of a computer is a complicated issue, a function of many interrelated quantities. These quantities include the application, the algorithm, the size of the problem, the high-level language, the implementation, the human level of effort used to optimize the program, the compiler's ability to optimize, the age of the compiler, the operating system, the architecture of the computer, and the hardware characteristics. The results presented for benchmark suites should not be extolled as measures of total system performance (unless enough analysis has been performed to indicate a reliable correlation of the benchmarks to the workload of interest) but, rather, as reference points for further evaluations.

Computational performance is often measured in terms of Megaflops, millions of floating point operations per second (Mflop/s). We usually include both additions and multiplications in the count of Mflop/s, and the reference to an

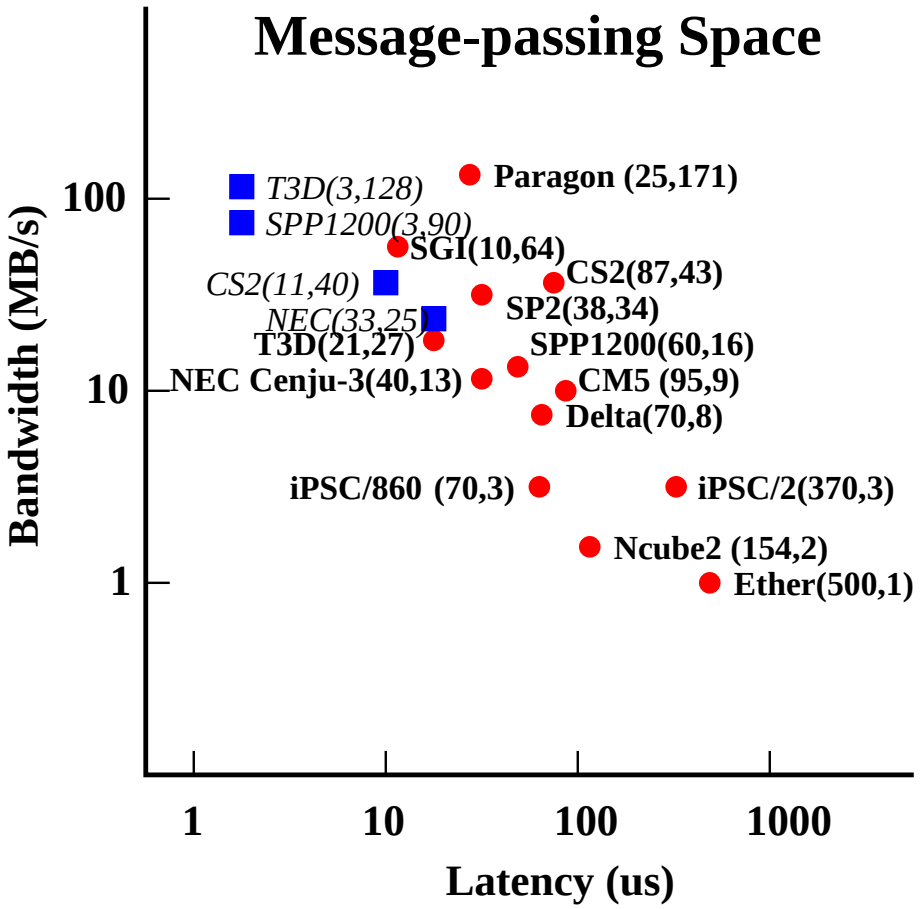

Figure 2.5: *Latency/bandwidth space for 0-byte message (latency) and 1 MB message (bandwidth). Block points represent shared-memory copy performance.*

operation is assumed to be on 64-bit operands.

The manufacturer usually refers to peak performance when describing a system. This peak performance is arrived at by counting the number of floating-point additions and multiplications that can be performed in a period of time, usually the cycle time of the machine. As an example, the IBM SP-1 processor, has a cycle time of 62.5 MHz. During a cycle the results of the multiply/add instruction can be completed giving:

$$2\ operations/1\ cycle\ *\ 1\ cycle/16nsec\ =\ 125\ Mflop/s.$$

Table 3.1 displays the performance for a single processor of various parallel computers using the LINPACK Benchmark [2]. The floating point execution rates have been converted to operations per cycle and also calculated the number of cycles consumed, as overhead (latency), during communication. At one time, a programmer had to go out of his way to code a matrix routine that would not run at nearly top efficiency on any system with an optimizing compiler. Owing to the proliferation of exotic computer architectures, this situation is no longer true.

Table 3.1: *Computation Performance.*

		Clock cycle		Linpack 100		Linpack 1000		Latency	
Machine	OS	MHz	(nsec)	Mfls	(ops/cl)	Mfls	(ops/cl)	us	(cl)
Convex SPP1000 (PVM)	SPP-UX 3.0.4.1	100	(10)	48	(.48)	123	(1.23)	76	(7600)
Convex SPP1000 (sm 1-n)								2.6	(260)
Convex SPP1000 (sm m-n)								11	(1080)
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	100	(8.33)	65	(.54)	123	(1.02)	63	(7560)
Convex SPP1200 (sm 1-n)								2.2	(264)
Convex SPP1200 (sm m-n)								11	(1260)
Cray T3D (sm)	MAX 1.2.0.2	150	(6.67)	38	(.25)	94	(.62)	3	(450)
Cray T3D (PVM)								21	(3150)
Intel Paragon	OSF 1.0.4	50	(20)	10	(.20)	34	(.68)	29	(1450)
Intel Paragon	SUNMOS 1.6.2							25	(1250)
Intel Delta	NX 3.3.10	40	(25)	9.8	(.25)	34	(.85)	77	(3080)
Intel iPSC/860	NX 3.3.2	40	(25)	9.8	(.25)	34	(.85)	65	(2600)
Intel iPSC/2	NX 3.3.2	16	(63)	.37	(.01)	–	(–)	370	(5920)
IBM SP-1	MPL	62.5	(16)	38	(.61)	104	(1.66)	270	(16875)
IBM SP-2	MPI	66	(15.15)	130	(1.97)	236	(3.58)	35	(2310)
KSR-1	OSF R1.2.2	40	(25)	15	(.38)	31	(.78)	73	(2920)
Meiko CS2 (MPI)	Solaris 2.3	90	(11.11)	24	(.27)	97	(1.08)	83	(7470)
Meiko CS2 (sm)								11	(990)
nCUBE 2	Vertex 2.0	20	(50)	.78	(.04)	2	(.10)	154	(3080)
nCUBE 1	Vertex 2.3	8	(125)	.10	(.01)	–	(–)	384	(3072)
NEC Cenju-3	Env Rev 1.5d	75	(13.3)	23	(.31)	39	(.52)	40	(3000)
NEC Cenju-3(sm)	Env Rev 1.5d	75	(13.3)	23	(.31)	39	(.52)	34	(2550)
SGI Power Challenge	IRIX 6.1	90	(11.11)	126	(1.4)	308	(3.42)	10	(900)
TMC CM-5	CMMD 2.0	32	(31.25)	–	(–)	–	(–)	95	(3040)

3.2. The LINPACK Benchmark

The LINPACK benchmark features solving a system of linear equation, $Ax = b$. The benchmark results examined here are for two distinct benchmark problems.

The first problem uses Fortran software from the LINPACK software package to solve a matrix problem of order 100. That is, the matrix A has 100 rows and columns and is said to be of size 100×100. The software used in this experiment is based on two routines from the LINPACK collection: DGEFA and DGESL. DGEFA performs the decomposition with partial pivoting, and DGESL uses that decomposition to solve the given system of linear equations. Most of the time - $O(n^3)$ floating-point operations - is spent in DGEFA. Once the matrix has been decomposed, DGESL is used to find the solution; this requires $O(n^2)$ floating-point operations.

DGEFA and DGESL in turn call three BLAS routines: DAXPY, IDAMAX, and DSCAL. For the size 100 benchmark, the BLAS used are written in Fortran. By far the major portion of time - over 90% at order 100 - is spent in subroutine DAXPY. DAXPY is used to multiply a scalar, α, times a vector, x, and add the results to another vector, y. It is called approximately $n^2/2$ times by DGEFA and $2n$ times by DGESL with vectors of varying length. The statement $y_i \leftarrow y_i + \alpha x_i$, which forms an element of the DAXPY operation, is executed approximately $n^3/3 + n^2$ times, which gives rise to roughly $(2/3)n^3$ floating-point operations in the solution. Thus, the benchmark requires roughly 2/3 million floating-point operations.

For the LINPACK 100 test, many processors achieve one floating point operation every four cycles, even though the process has the ability to deliver much more than this. The primary reason for this lack of performance relates to the poor compiler generated code and the algorithm's ineffective use of the memory hierarchy. There are a few exceptions, most notably the IBM SP-2's processor. The RS/6000-590 processor is able to achieve two floating point operations per cycle for the LINPACK 100 test, because the compiler and the cache structure work together. There are also examples of poor performance on some of the first generation parallel machines, such as the nCUBE 1 and 2 and the Intel iPSC/2. These processors are able to achieve only .01 to .04 floating point operations per cycle.

In the second benchmark, the problem size is larger (matrix of order 1000), and modifying or replacing the algorithm and software is permitted to achieve as high an execution rate as possible. (The optimized programs must still maintain the same relative accuracy as standard techniques.) The algorithm used for the $n = 1000$ problem makes better use of the memory hierarchy by utilizing the data in cache. Thus, the hardware had more opportunity for reaching near-asymptotic rates. Most of the processors achieve 70 to 80 % of their peak.

If we examine the algorithm used in LINPACK and look at how the data

are referenced, we see that at each step of the factorization process there are operations that modify a full submatrix of data. This update causes a block of data to be read, updated, and written back to central memory. The number of floating-point operations is $(2/3)n^3$, and the number of data references, both loads and stores, is $(2/3)n^3$. Thus, for every *add/multiply* pair we must perform a load and store of the elements, unfortunately obtaining no reuse of data. Even though the operations are fully vectorized, there is a significant bottleneck in data movement, resulting in poor performance. To achieve high-performance rates, this *operation-to-memory-reference rate* must be higher.

Just as the operation-to-memory-reference rate affects performance at the instruction level, the computation-to-communication ratio for message passing will affect application performance at a coarser level. The compiler and/or application programmer can improve application performance by algorithm restructing to increase data re-use and by reducing or pipelining message passing.

4. Future comparisons

In order that the reader might perform these same tests, both LINPACK and the message-passing tests are available over the Internet.

4.1. Rules for Running the Tests

The message-passing test software intentionally has been kept simple so that it will be easy for an experienced programmer to adapt the program, or parts of it, to a specific architecture with only a modest effort. In running the tests, the user is allowed to change the message passing calls to the appropriate call on the specific system the program is to be run on. We have provided both PVM and MPI [4] implementations in netlib.

4.2. Obtaining the Software

The software used to generate the data for this report can be obtained by sending electronic mail to *netlib@www.netlib.org* . To receive the single-precision software for this benchmark, in the mail message to *netlib@www.netlib.org* type *send comm.shar from benchmark* . To receive the double-precision software for this benchmark, type *send comm.shar from benchmark* .

A web browser can be used as well. With the url

http://www.netlib.org/benchmark/index.html

click on "benchmark/comm.shar".

5. References

[1] HPCwire No. 4912 12/2/94, 1994. Email exchange.

[2] J. Dongarra. Performance of various computers using standard linear equations software in a Fortran environment. Technical Report CS-89-85, University of Tennessee, 1995.

[3] T. H. Dunigan. Early experiences and performance of the intel paragon. Technical report, Oak Ridge National Laboratory, 1993. ORNL/TM-12194.

[4] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard . *International Journal of Supercomputer Applications and High Perf ormance Computing*, 8(3/4), 1994. Special issue on MPI. Also available electronically, the url is `ftp://www.netlib.org/mpi/mpi-report.ps`.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[6] Roger Hockney. The communication challenge for mpp. *Parallel Computing*, 20:389–398, 1994.

Appendix

A. Appendix: Machine Configurations for Echo Tests

A summary of the various architectures and configurations used when these performance figures were measured follows. Unless otherwise noted, the test programs were compiled with *cc -O*.

The **Convex SPP1000** and SPP1200 consist of SCI-ring connected nodes (160 MB/second). Each SPP1000 node consists of eight 100 MHz HP PA RISC 7100 processors (120 MHz for the SPP1200) with a cross-bar memory interconnect (250 MB/second). The tests were run under SPP-UX 3.0.4.1 and ConvexPVM 3.3.7.1.

The **Cray T3D** is 3-D-torus multiprocessor using the 150 MHz DEC Alpha processor. Communication channels have a peak rate of 300 MB/second. Tests were performed using MAX 1.2.0.2. A special thanks to Majed Sidani of Cray for running our communication tests on the T3D using PVM. The PVM communication was with pvm_psend and pvm_precv.

The **Intel iPSC/860** is Intel's third generation hypercube. Each node has a 40 MHz i860 with 8 KB cache and at least 8 MB of memory. Communication channels have a peak rate of 2.8 MB/second. Tests were performed using NX 3.3.2. The **Intel iPSC/2** uses the same communication hardware as the iPSC/860 but uses 16 MHz 80386/7 for computation.

The **Intel Delta** is a 512-node mesh designed as a prototype for the Intel Paragon family. Each node has a 40 MHz i860 with 8 KB cache and 16 MB of memory. Communication channels have a peak rate of 22 MB/second. Tests were performed using NX 3.3.10.

The **Intel Paragon** is a mesh-based multiprocessor. Each node has at least two 50 MHz i860XP processors with 16 KB cache and at least 16 MB of memory. One processor is usually dedicated to communications. Communication channels have a peak rate of 175 MB/second. Test were run under OSF 1.0.4 Server 1.3/WW48-02 and SUNMOS 1.6.2 (using NX message passing).

The **IBM SP1** is an omega-switch-based multiprocessor using 62.5 MHz RS6000 processors. Communication channels have a peak rate of 40 MB/second. Tests were run using MPL.

The **IBM SP2** is an omega-switch-based multiprocessor using 66 MHz RS6000 processors with L2 cache. Communication channels have a peak rate of 40 MB/second. Tests were run using MPI. The MPI communication was with mpi_send and mpi_recv.

The **Kendall Square** architecture is a shared-memory system based on a hierarchy of rings using a custom 20 MHz processor. Shared-memory latency is about 7 μs, and bandwidth is about 32 MB/second. The message-passing performance was measured using Pacific Northwest Laboratory's *tcgmsg* library on one ring of a KSR1 running OSF R1.2.2.

The **Meiko CS2** uses SPARC processors with 200 Mflop/s vector co-processors. The communication topology is a fat tree with peak bandwidth of 50 MB/second. The MPSC message-passing library was used for the echo tests. Meiko notes that using point-to-point bidirectional channels in the echo test reduces latency from 82 microseconds to 14 microseconds. A special thanks to Jim Cownie of Meiko for running our communication tests.

The **Ncube** hypercube processors are custom processors with hypercube communication integrated into the chip. The first generation chip ran at 8 MHz, the second generation chip ran at 20 MHz.

The **NEC Cenju-3** results are from a 75 MHz VR4400SC MIPS processor with 32 KBytes of primary cache and 1 MByte of secondary cache using MPI under the Cenju Environment Release 1.5d. Communication channels have a peak rate of 40 MB/second through a multistage interconnection network.

The **SGI** results are from a 90 MHz PowerChallenge using MPI under IRIX 6.1. The SGI is a shared-memory multiprocessor using a 1.2 GB/s bus.

The **TMC CM5** is hypertree multiprocessor using 32 MHz SPARC processors with four vector units and 16 MB of memory per node. Communication channels have a peak rate of 20 MB/second. Tests were run using the message passing library CMMD 2.0.